October, 2001

KEY ESTABLISHMENT SCHEMES WORKSHOP DOCUMENT:

(November 1-2, 2001)

[Note to the reader: This workshop document is a working paper from which material will be drawn for the schemes document. This workshop document contains the current thinking of the working group that will develop the final schemes document. Editorial comments are not appropriate at this time, but comments addressing its content are encouraged. Questions have been inserted to engender thought as to the eventual content of the schemes document. The current content and the inserted questions will be addressed at the Key Management workshop.]

Table of Contents

Table of Figures

Table of Tables

KEY ESTABLISHMENT SCHEMES APPROACH

1. Introduction

Many U.S. Government Information Technology (IT) systems need to employ well-established cryptographic schemes to protect the integrity and confidentiality of the data that they process. Algorithms such as the Advanced Encryption Standard (AES) as defined in Federal Information Processing Standard (FIPS) 197, Triple DES as adopted in FIPS 46-3, and HMAC as defined in FIPS 198 make attractive choices for the provision of these services. These algorithms have been standardized to facilitate interoperability between systems. However, the use of these algorithms requires the establishment of shared keying material in advance. Trusted couriers may manually distribute this keying material. However, as the number of entities using a system grows, the work involved in the distribution of the keying material could grow exponentially. Therefore, it is essential to support the cryptographic algorithms used in modern U.S. Government applications with automated key establishment schemes.

2. Scope and Purpose

This workshop document provides an approach for the development of a key establishment schemes document (hereafter referred to as the final schemes document) from standards developed by the American National Standards Institute (ANSI): ANSI X9.42, *Agreement of Symmetric Keys using Discrete Logarithm Cryptography*, and ANSI X9.63, *Key Agreement and Key Transport using Elliptic Curve Cryptography*. It is intended that the final schemes document will also contain key transport schemes from ANSI X9.44, *Key Agreement and Key Transport using Factoring-Based Cryptography;* a key wrapping technique, whereby a symmetric key is encrypted using another symmetric key (e.g., an AES key is encrypted by an AES key); and a discussion of keys derived from a "master key".

[Note: The key transport schemes from ANSI X9.44, the key wrapping schemes, and keys derived from a master key are not present in this workshop document due to time constraints. They will be provided for review at a later time.]

This workshop document provides only a high level description of the schemes defined in the ANSI standards. Details for the implementation of these schemes, including the detailed instructions for proper implementation, are available in the appropriate ANSI standard. When there are differences between this workshop document and the referenced ANSI standards, this workshop document identifies those differences.

The final schemes document will contain a requirement that the schemes be used in conjunction with Special Publication 800-X, *Guideline for Key Management and Use of Cryptographic Mechanisms* [8]. The schemes document, the referenced ANSI standards, and the guideline [8] are intended to provide sufficient information for a vendor to implement and test secure key establishment for FIPS 140-2 [1] validated modules.

3. Definitions, Symbols and Abbreviations

3.1 Definitions

Approved	FIPS approved or NIST Recommended.
Keying material	The data (e.g., keys and IVs) that are necessary to establish and maintain cryptographic keying relationships. As used in this workshop document, keying material is established between participants in a key establishment process.
Shared keying material	The keying material that is derived by applying the key derivation function of Section 6.3 to the shared secret.
Shared secret	A secret value that has been computed using a prescribed algorithm and combination of keys belonging to the participants in the key establishment scheme. The shared secret **must** not be used directly as shared keying material.

3.2 Symbols and Abbreviations

General:

H	An Approved hash function.
$Text_1$, $Text_2$	An optional bit string that may be used during key confirmation and that is sent between the parties establishing keying material.
U	One entity of a key establishment process, or the bit string denoting the identity of that entity.
V	The other entity of a key establishment process, or the bit string denoting the identity of that entity.
$X\|\|Y$	Concatenation of two strings X and Y.

The following notations are consistent with that used in the ANSI standards; however, it should be recognized that the notation between the standards is inconsistent (e.g., x and y are used as the private and public keys in ANSI X9.42, whereas x and y are used as the coordinates of a point in ANSI X9.63).

ANSI X9.42:

p, q, g	The domain parameters.
mod p	The reduction modulo p on an integer value.

$r_{U,}$ r_V	Party U or Party V's ephemeral private key.
$t_{U,}$ t_V	Party U or Party V's ephemeral public key.
$x_{U,}$ x_V	Party U or Party V's static private key.
$y_{U,}$ y_V	Party U or Party V's static public key.
Z	A shared secret that is used to derive keying material using a key derivation function.
Z_e	An ephemeral shared secret that is computed using a Diffie-Hellman primitive.
Z_s	A static shared secret that is computed using a Diffie-Hellman primitive.

ANSI X9.63:

[*X*]	Indicates that the inclusion of string *X* is optional.
a, *b*	Field elements that define the equation of an elliptic curve.
avf(*P*)	The associate value of the elliptic curve point *P*.
$d_{e,U}$, $d_{e,V}$	Party U's and Party V's ephemeral private keys.
$d_{s,U}$, $d_{s,V}$	Party U's and Party V's static private keys.
FR	An indication of the basis used.
G	A distinguished point on an elliptic curve.
h	The order of the elliptic curve divided by the order of the point *G*. This is called the cofactor.
n	The order of the point *G*.
j	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group.
q	The field size.
$Q_{e,U}$, $Q_{e,V}$	Party U's and Party V's ephemeral public keys.
$Q_{s,U}$, $Q_{s,V}$	Party U's and Party V's static public keys.
SEED	An optional bit string that is present if the elliptic curve was randomly generated.

x_P	The x-coordinate of a point P.
y_P	The y-coordinate of a point P.
Z	A shared secret that is used to derive key using a key derivation function.
Z_e	An ephemeral shared secret that is computed using a Diffie-Hellman primitive.
Z_s	A static shared secret that is computed using a Diffie-Hellman primitive.

4. Key Establishment Algorithm Classes

Cryptographic keying material may be electronically established between parties using either key agreement or key transport schemes. During key agreement, the keying material to be established is not sent; information is exchanged between the parties that allows the calculation of the keying material. The key agreement schemes described in this workshop document use asymmetric (public key) techniques. During key transport, encrypted keying material is sent from an initiator who generates the keying material to another party. Key transport schemes use either symmetric or public key techniques.

The schemes from ANSI X9.42 and X9.63 in this workshop document are based on the intractability of the discrete logarithm problem. The schemes in ANSI X9.42 are calculated over a finite field. The schemes specified in ANSI X9.63 are calculated using elliptic curves.

This workshop document includes a high-level specification of the key establishment schemes. Implementation details (e.g., data conversion rules, arithmetic, basis, encoding rules, etc.) are provided in the appropriate ANSI standard. When there are differences between this workshop document and the referenced ANSI standards, the final schemes document will contain a requirement that the schemes document will have precedence for U.S. Government applications.

5. Security Attributes

[The final schemes document will contain a section that describes the security attributes that may be provided by the various key establishment schemes and will discuss the importance of each attribute.]

6. Cryptographic Elements

This workshop document assumes that the reader has, and is familiar with, ANSI X9.42 and ANSI X9.63. These standards should be consulted to obtain specific guidance.

6.1 Domain Parameters

Discrete Log based cryptography, as specified in ANSI X9.42 and X9.63, requires that the public and private key pairs be generated with respect to domain parameters. These domain parameters must be validated to ensure that the parameters have been generated properly (e.g., by a

Certification Authority, hereafter called a CA). Although domain parameters are public information, they must be managed so that the correct correspondence between a given key pair and its domain parameters is maintained for all parties that use the key pair. Domain parameters may remain fixed for an extended time period and may be used with multiple key establishment schemes.

Some of the schemes in ANSI X9.42 and X9.63 allow separate domain parameters to be used with static and ephemeral keys in the same scheme. **This workshop document, however, uses only one set of domain parameters, i.e., the same domain parameters are used with the static and ephemeral keys in any given scheme.**

6.1.1 Domain Parameter Generation

Domain parameters for the schemes specified in ANSI X9.42 are of the form *(p, q, g)*, where *p* and *q* are prime, and *g* is generator of the *q*-order cyclic subgroup of *GF(p)**. For specific instructions on how to generate *(p, q, g)* see Section 6.1 of ANSI X9.42 [9].

Domain parameters for the schemes specified in ANSI X9.63 are of the form (*q, FR, a, b, [SEED], G, n, h*), where *q* is the field size; *FR* is an indication of the basis used; *a* and *b* are two field elements that define the equation of the curve; *SEED* is an optional bit string if the elliptic curve was randomly generated in a verifiable fashion; *G* is a point, (x_G, y_G) of prime order on the curve; *n* is the order of the point *G*; and *h* is the cofactor equal to the order of the curve divided by *n*. The generation of domain parameters depends on the form of the field over which the curve is defined. For specific instructions on how to generate *(q, FR, a, b, [SEED], G, n, h)*, see Section 5.1.1.1 of [ANSI X9.63] for curves over F_p, and Section 5.1.2.1 of [ANSI X9.63] for curves over F_{2^m}. Recommended elliptic curves for the Federal Government have been provided in FIPS 186-2 [3] that may be used to determine the domain parameters.

6.1.2 Domain Parameter Validation

Domain parameters **must** be validated by each party or by an entity that they trust before use. Validation consists of verifying the arithmetic properties of the domain parameters. Each party **must** obtain assurance that the domain parameters are valid in one of the following ways:

i. The party has generated the parameters. Note: the checking of the parameters is actually done during generation.
ii. The party has performed the domain parameter validation as specified in:
    - Section 7.2 of ANSI X9.42 for discrete logarithms,
    - Section 5.1.1.2 of ANSI X9.63 for elliptic curves over F_p, or
    - Section 5.1.2.2 of ANSI X9.63 for elliptic curves over F_{2^m}.
iii. The party has received assurance from a trusted party (e.g., a CA) that the parameters are valid (either (i) or (ii) above is true for the trusted party that provides the assurance).

6.1.3 Domain Parameter Management

Only authorized (trusted) parties should generate domain parameters. In addition, each private-public key pair must be correctly associated with its domain parameters (e.g., by using a public

key certificate). The unauthorized modification or substitution of domain parameters may cause security risks.

6.2 Private/Public Keys

6.2.1 Private/Public Key Generation

Static and ephemeral key pairs are generated using the same primitives.

For the schemes specified in ANSI X9.42, generate a private key *x* and a public key *y* using the domain parameters (*p, q, g*). See Section 7.3 of [9].

For the schemes specified in ANSI X9.63, generate a private key *d* and a public key *Q* using the domain parameters (*q, FR, a, b, [SEED], G, n, h*). See Section 5.2.1 of [11].

When static public/private keys are required by a participant in a key establishment process, the static keys **must** be generated prior to participation in the key establishment process. Each private key **must** be statistically unique, unpredictable, and created using an Approved random number generator. The same private key should not be used with more than one set of domain parameters.

6.2.2 Public Key Validation

Secure key establishment depends on the validity of the public/private key pairs. This workshop document **requires** public key validation to be performed by the receiver of a key.

Static public keys **must** be validated by the recipient, or by an entity that is trusted by the recipient. Static public key validation may be accomplished in one of the following three ways; the first two options are preferred:

1. Performing an explicit public key validation, i.e., checking that the public key has certain mathematical properties (i.e., the key is *reasonable*; this does not determine that the other party has a private key that is associated with this public key). Note: this is method 1 in ANSI X9.42 (Section 7.4) and ANSI X9.63 (Section 5.2.2).
2. Receiving assurance that another party (e.g., a CA) has validated the public key by using method 1. Note: this is defined as method 2 in ANSI X9.42 (Section 7.4), and as method 3 in ANSI X9.63 (Section 5.2.2).
3. Receiving assurance that another party (e.g., a CA) has regenerated the public key using trusted routines. Note: this is method 4 in ANSI X9.42 (Section 7.4) and ANSI X9.63 (Section 5.2.2).

Each ephemeral public key **must** be validated by the recipient before being used to derive a shared secret. Ephemeral public keys may be validated using either method 1 or method 2 above.

Note: Both of the ANSI standards include an additional option for public key validation. In this option, an implicit public key validation is performed by generating the public key using trusted routines (method 3 of ANSI X9.42 (Section 7.4) and method 2 of ANSI X9.63 (Section 5.2.2)). The working group did not see a reason for including this method. Can anyone provide a reason?

6.2.3 Key Pair Management

Public/private key pairs **must** be correctly associated with their corresponding domain parameters. Private keys **must** be protected from unauthorized disclosure, modification, and substitution. Thus, they **must** be protected from any unauthorized access.

The cryptoperiod of each static private key **must** be clearly defined. Static private keys **must** be destroyed at the end of their cryptoperiod. Ephemeral keys should be generated as close to their time of use as possible. Ephemeral private keys **must** be destroyed immediately after the shared secret is computed.

Recipients of static public keys **must** be assured of a binding between the public key, a set of domain parameters and the entity that "owns" the keys (i.e., the entity with whom the recipient intends to establish a key). This assurance is often provided by verifying a public-key certificate signed by a trusted party (e.g., a Certification Authority).

Static public keys **must** be obtained in a trusted manner, e.g., from a certificate signed by a trusted CA, or directly from the public key owner, provided that the public key owner is trusted by the receiving party and can be authenticated as the source of the data that is received.

Static key pairs may be used by more than one key establishment scheme. However, different public/private key pairs should be used for different purposes (e.g., digital signature key pairs should not be used for key establishment).

6.3 Key Derivation Function

The key derivation function is used to derive keying material from a shared secret as follows:

Function call: *kdf*(*Z,OtherInput*), where *OtherInput* is *U, V, keydatalen, hashlen,* [*SharedInfo*].

Input description:

1. A bit string *Z* that is the shared secret.
2. Bit strings *U* and *V* that denote the identities of the participating parties, where *U* is the party that initiated the key establishment process, and *V* is the responder in the key establishment process.
3. An integer *keydatalen* that is the length in bits of the keying material to be generated. *keydatalen* must be less than $hashlen \times (2^{32}-1)$.
4. An integer *hashlen* that is the length in bits of the hash function to be used to derive the keying material.
5. An optional bit string *SharedInfo* that consists of data shared by parties U and V.

Process:

a. Initiate a 32-bit, big-endian bit string *counter* as 00000001_{16}.

b. For i=1 to $i=\lceil keydatalen / hashlen \rceil$, do the following:

   Compute $Hash_i = H(Z||counter||U||V||[SharedInfo])$
   Increment *counter*
   Increment *i*.

c. Let $Hhash_j$ denote $Hash_j$ if *keydatalen/hashlen* is an integer, and let it denote the (*keydatalen*-(*hashlen*×(*j-1*))) leftmost bits of $Hash_j$ otherwise.

d. Set *DerivedKeyingMaterial* = $Hash_1||Hash_2||\ldots||Hash_{j-1}||Hhash_j$.

Output: The bit string *DerivedKeyingMaterial* of *keydatalen* bits.

[Question: Since both parties need to compute the same derived keying material, does the necessity to provide U and V create a problem? For example, can all applications determine who the initiator and responder are so that they would know in what order to place the identities? Or would it be better to alphabetize the identities? Rather than using "U||V", would it be better to use a function of the two entities, such as "(U OR V)||(U AND V)", where "OR" and "AND" are bit-wise logical operators?]

The derived keying material may be parsed into one or more keys or other cryptographic keying material (e.g., IVs).

Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to $hashlen \times (2^{32}-1)$ bits **must** output "invalid" and stop.

Note: The security provided by the derived keying material is limited by the cryptographic strength of the key establishment scheme. For example, if an elliptic curve scheme is used with a 160 bit private key, a strength of roughly 80 bits is provided to the derived keying material, i.e., a 128 bit AES key derived from such a scheme provides no more than 80 bits of security to the information it protects, not the 128 bits of security that might be expected when using AES with a truly random key.

Note: This technique differs from the key derivation functions defined in ANSI X9.42 and X9.63. In this workshop document, the identities of the initiating party (U) and the recipient (V) are included in the data that is hashed. In ANSI X9.42 and X9.63, these identities could be considered a part of the *SharedInfo*, which is optional and not defined. ANSI X9.42 also includes a second key derivation function that is based on ASN.1 DER encoding.

6.4 MAC

A Message Authentication Code (MAC) is a function of both a symmetric key and other data. In key establishment schemes, an entity is sometimes required to compute a *MacTag* on received or derived data, which is sent to the other entity in order to confirm that the data was correctly received or derived. This workshop document **requires** that an Approved MAC algorithm be used to compute a MAC.

The MAC function is used to provide key confirmation, when desired, and is used to validate implementations of the key establishment schemes specified in this workshop document. Tag computation and checking are defined in Section 7.8 of ANSI X9.42, and in Section 5.7 of ANSI X9.63.

6.4.1 Tag computation

The computation of the Tag is represented as follows:

$$MacTag = MAC_{MacKey}(MacData).$$

MAC represents an Approved MAC, *MacKey* represents a symmetric key, and *MacData* represents the data.

6.4.2 Tag Checking

To check a *MacTag* for given *MacKey* and *MacData*, the *MacTag* is computed by the receiver using the received or derived *MacData* (as specified in Section 6.4.1) and compared with the received *MacTag*. If the two *MacTag* values are equal, then it may be inferred that the *MacKey* and *MacData* values computed by each party are equal.

6.4.3 Implementation validation

For purposes of validating an implementation of the schemes in this workshop document during an implementation validation test, the value of *MacData* **must** be the string "Standard Test Message" followed by 16 bytes containing a 128-bit value used as a nonce. The default value for the nonce is all zeros. Different values of the nonce will be specified during testing to perform known answer tests.

Note: ANSI X9.42 defines MacData as "ANSI X9.42 Testing Message". ANSI X9.63 does not address implementation validation at this level of detail.

6.5 Associate Value Function (Elliptic Curve Only)

The associate value function is used in ANSI X9.63 by the MQV family of key agreement schemes to compute an integer associated with an elliptic curve point. This workshop document defines *avf*(*P*) to be the associate value function of a point *P* ($P \neq j$) as defined in Section 5.6.1 of ANSI X9.63 using the domain parameters (*q, FR a, b, [SEED], G, n, h*).

6.6 Cryptographic Hash Functions

An Approved hash function **must** be used when a hash function is required (e.g., for the key derivation function or to compute a MAC when HMAC is used).

6.7 Random Number Generation

Whenever this standard requires the use of a randomly generated value (e.g., for keys and nonces), the values **must** be generated using an Approved random number generator.

6.8 Key Confirmation

Key confirmation is used to provide assurance that the parties have derived the same keys. The participants in the key establishment process must first establish a shared secret *Z* using one of the schemes provided by this standard. Each entity **must** then derive keying material using Section 6.3 and parse it into *MacKey* and *KeyData*, such that:

$$MacKey \,||\, KeyData = DerivedKeyingMaterial$$

The responder in the key establishment process (party *V*, for example) sets $MacData_V = 02_{16}||V||U||EphemPubKeyV||EphemPubKeyU||[Text_1]$ and computes $MacTag_V$ (see Section 6.4.1), which is then sent to the initiator (party *U* in this example).

The initiator in the key establishment process (party *U*, for example) sets $MacData_U = 03_{16}||U||V||EphemPubKeyU||EphemPubKeyV||[Text_2]$ and computes $MacTag_U$ (see Section 6.4.1), which is then sent to the responder (party *V* in this example)..

Note that *MacData* need not be sent if both parties know all information needed to construct it.

Both parties verify the received *MacTag* using the Tag Checking procedure defined in Section 6.4.2.

Note: Key Confirmation is defined for some of the schemes in ANSI X9.63. Key Confirmation for ANSI X9.42 is addressed in a different ANSI standard, ANSI X9.70, *Management of Symmetric Keys Using Public Key Cryptography*.

Note: The above specification is restricted to those schemes where both parties have ephemeral keys. Further research is needed to determine whether key confirmation is also worth doing for other schemes and how best to do it.

[Questions:

1. Which schemes should use key confirmation? In X9.63, addresses key confirmation for a Combined Unified Model (a combination of the Ephemeral Unified Model and the Static Unified Model), the Full Unified Model, and the Full MQV scheme.
2. Should key confirmation be mandatory for some schemes?
3. Can all applications determine who the initiator and responder are?

]

6.9 Calculation of Shared Secrets

Primitives for the calculation of the shared secrets are defined in the ANSI standards. Each key establishment scheme is required to use exactly one primitive. The five ANSI primitives that shall be used by the schemes in Section 7 of this workshop document are:

1. The Diffie-Hellman primitive of Section 7.5.1 in ANSI X9.42. This primitive must be used by the dhHybrid1, dhEphem, dhHybridOneFlow, dhOneFlow and dhStatic schemes, which are based on finite field arithmetic and the Diffie-Hellman algorithm.
2. The Modified Diffie-Hellman primitive of Section 5.4.2 of ANSI X9.63. This primitive must be used by the Full Unified Model, Ephemeral Unified Model, 1-Pass Unified Model, 1-Pass Diffie-Hellman and Static Unified Model schemes, which are based on elliptic curve arithmetic and the Diffie-Hellman algorithm.
   Note: This differs from ANSI X9.63 in the use of primitives for elliptic curve-based Diffie-Hellman schemes. ANSI X9.63 allows a choice of either the Standard Diffie-Hellman primitive (which does not use co-factor multiplication) or the Modified Diffie-Hellman primitive (which uses co-factor multiplication), while this workshop document requires the use of the Modified Diffie-Hellman primitive for elliptic curve-based Diffie-Hellman schemes.
3. The MQV2 primitive of Section 7.5.2.1 of ANSI X9.42. This primitive must be used by the MQV2 interactive scheme, which is based on finite field arithmetic and the MQV algorithm.

4. The MQV1 primitive of Section 7.5.2.2 of ANSI X9.42. This primitive must be used by the MQV1 store-and-forward scheme, which is based on finite field arithmetic and the MQV algorithm.
5. The MQV primitive of Section 5.5 of ANSI X9.63. This primitive must be used by the Full MQV and 1-Pass MQV schemes, which are based on elliptic curve arithmetic and the MQV algorithm.

Shared secrets **must not** be used directly as shared keying material; the shared keying material **must** be calculated by applying the key derivation function to the shared secret (see Section 6.3 of this workshop document).

6.10 RSA Primitives

[To be addressed later]

6.11 Key Wrapping Primitive(s)

[To be included later]

7. Key Agreement Schemes

This workshop document provides three **categories** of key establishment schemes (See Table 1). The classification of the categories is based on the number of ephemeral keys used by the two parties to the key agreement process, U and V. In category *C(i)*, parties U and V have a total of *i* ephemeral key pairs. The first category, *C(2)* consists of schemes requiring the generation of ephemeral key pairs by both parties (two party participation schemes - i.e. interactive or 2-way schemes); the second category, *C(1)* consists of schemes requiring the generation of an ephemeral key pair by only one party (one party participation schemes - i.e., store and forward or 1-way schemes); and the third category, *C(0)* consists of schemes that do not use ephemeral keys (static schemes - i.e., passive schemes).

Table 1: Key Agreement Scheme Categories

Category	Comment
C(2): Two party participation (interactive, 2-way)	Each party generates an ephemeral key pair.
C(1): One party participation (store and forward, 1-way)	Only the initiator generates an ephemeral key pair.
C(0): Static (passive)	No ephemeral keys are used.

Each category is comprised of one or more subcategories that are classified by the use of static keys by the parties (see Table 2). In subcategory *C(i,j)*, parties U and V have a total of *i* ephemeral key pairs and *j* static key pairs.

Table 2: Key Agreement Scheme Subcategories

Category	Subcategory
C(2): Two party participation (interactive, 2-way)	*C(2,2):* Each party generates an ephemeral key pair and has a static key pair.
	C(2,0): Each party generates an ephemeral key pair; no static keys are used.

C(1): One party participation (store and forward, 1-way)	*C(1,2):* The initiator generates an ephemeral key pair and has a static key pair; the responder has only a static key pair.
	C(1,1): The initiator generates an ephemeral key pair, but has no static key pair; the responder has only a static key pair.
C(0): Static (passive)	*C(0,2):* Each party has only static keys.

The schemes may be further classified by whether they use finite field arithmetic (FF) as specified in ANSI X9.42 or elliptic curve arithmetic (EC) as specified in ANSI X9.63. A scheme may use either Diffie-Hellman (DH) or MQV primitives (see Section 6.9). Thus, for example, C(2,2,DH,FF) completely classifies the dhHybrid1 scheme as a scheme with two ephemeral keys and two static keys that uses a Diffie-Hellman primitive with finite field arithmetic (See Table 3).

Table 3 : Key Agreement Schemes

Category	Subcategory	Primitive	Arithmetic	Scheme	Full Classification
C(2)	*C(2,2)*	DH	FF	dhHybrid1	C(2,2,DH,FF)
C(2)	*C(2,2)*	DH	EC	Full Unified Model	C(2,2,DH,EC)
C(2)	*C(2,2)*	MQV	FF	MQV2	C(2,2,MQV,FF)
C(2)	*C(2,2)*	MQV	EC	Full MQV	C(2,2,MQV,EC)
C(2)	*C(2,0)*	DH	FF	dhEphem	C(2,0,DH,FF)
C(2)	*C(2,0)*	DH	EC	Ephemeral Unified Model	C(2,0,DH,EC)
C(1)	*C(1,2)*	DH	FF	dhHybridOneFlow	C(1,2,DH,FF)
C(1)	*C(1,2)*	DH	EC	1-Pass Unified Model	C(1,2,DH,EC)
C(1)	*C(1,2)*	MQV	FF	MQV1	C(1,2,MQV,FF)
C(1)	*C(1,2)*	MQV	EC	1-Pass MQV	C(1,2,MQV,EC)
C(1)	*C(1,1)*	DH	FF	dhOneFlow	C(1,1,DH,FF)
C(1)	*C(1,1)*	DH	EC	1-Pass Diffie-Hellman	C(1,1,DH,EC)
C(0)	*C(0,2)*	DH	FF	dhStatic	C(0,2,DH,FF)
C(0)	*C(0,2)*	DH	EC	Static Unified Model	C(0,2,DH,EC)

Each party in a key agreement process **must** use the same domain parameters. These parameters **must** be established prior to the initiation of the key agreement process. See Section 6.1 for a discussion of domain parameters.

A general flow diagram is provided for each category of schemes. The dotted-line arrows represent the distribution of static public keys that may be distributed by the parties themselves or by a third party, such as a Certification Authority (CA). Static public keys **must** be distributed in a trusted manner such that each party's static public key is bound to that party's identity and to a set of domain parameters, e.g., by a public key certificate signed by a trusted CA. The binding process **must** include the validation of the static public key as discussed in Section 6.2.2. The

acquisition of the static public keys may take place any time prior to their use, either before or during the key agreement process.

The solid-line arrows represent the distribution of ephemeral public keys that occur during the key agreement process. Ephemeral key pairs **must** be generated as close to the time of use as is possible, and **must** be destroyed as soon as the computational need for them is complete. Ephemeral public keys **must** be validated as specified in Section 6.2.2.

7.1 Two Party Participation (interactive, 2-way), C(2)

In this category, each party generates an ephemeral key pair and sends the ephemeral public key to the other party. The two parties perform similar computations to derive their shared secret; however, the key derivation calculation (see Section 6.3) and the key confirmation calculation (if used - see Section 6.8) differ for the initiator and responder. In this situation, the scheme descriptions should be interpreted with U designating the initiator and V designating the responder.

This category consists of two subcategories that are determined by the use of static keys by the parties. In the first subcategory, each party has both static and ephemeral keys (see Section 7.1.1), while in the second subcategory, each party has only ephemeral keys (see Section 7.1.2).

7.1.1 Each party has a static key pair and generates an ephemeral key pair: C(2,2)

For these schemes, each party (U and V) have static key pairs and generate ephemeral key pairs during the key agreement process. All key pairs **must** be generated using the same domain parameters. Party U and party V obtain each other's static keys, which have been generated prior to the key establishment process. Both parties generate ephemeral private/public key pairs and exchange the ephemeral public keys. Using the static and ephemeral keys, both parties generate a shared secret. The shared keying material is derived from the shared secret (see Figure 1).

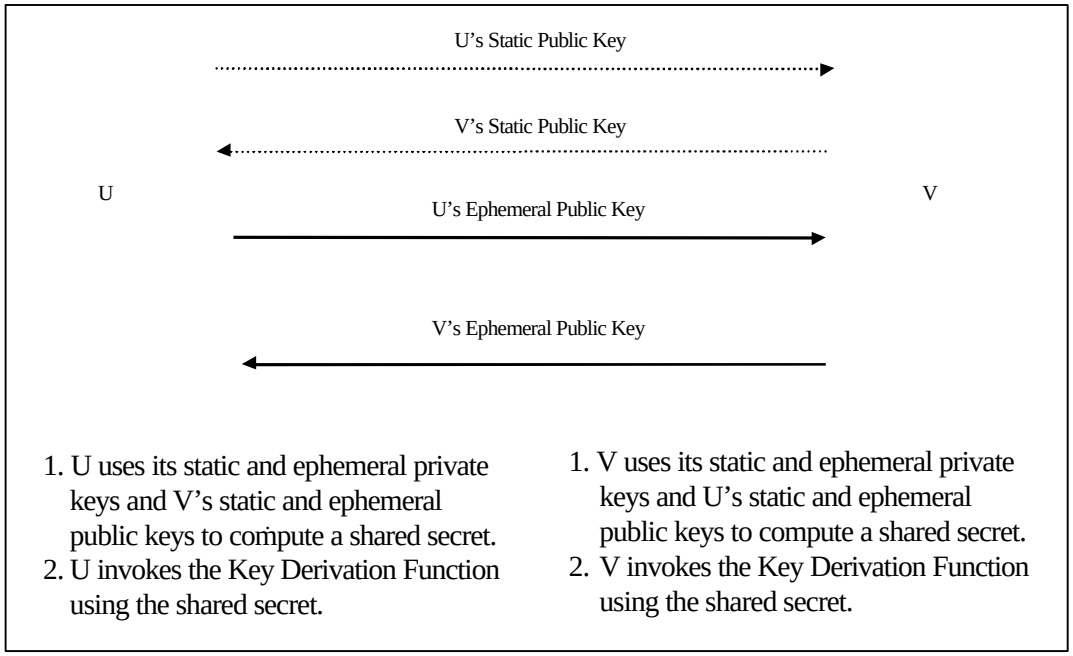

Figure 1: General protocol when each party has both static and ephemeral key pairs

7.1.1.1 dhHybrid1, C(2,2,DH,FF)

In this scheme from ANSI X9.42, each party has a static key pair (x, y) that was previously generated as specified in Section 6.2.1 using the same domain parameters (p, q, g). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, each party generates an ephemeral key pair (r, t) using the same domain parameters (p, q, g) that were used to generate the static key pair and sends the ephemeral public key t to the other party. Party U generates (r_U, t_U) and sends t_U to party V; party V generates (r_V, t_V) and sends t_V to party U. Each party computes the shared secret Z as shown in Table 4, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 4: ANSI X9.42 dhHybrid1 Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), x_U, y_V, r_U, t_V	(p, q, g), x_V, y_U, r_V, t_U

Computation	$Z_s = y_V^{x_U} \bmod p$ $Z_e = t_V^{r_U} \bmod p$	$Z_s = y_U^{x_V} \bmod p$ $Z_e = t_U^{r_V} \bmod p$
Derive Key Material	Compute *kdf*(*Z,OtherInput*) using $Z = Z_e \| \| Z_s$	Compute *kdf*(*Z,OtherInput*)using $Z = Z_e \| \| Z_s$

7.1.1.2 Full Unified Model, C(2,2,DH,EC)

In this scheme from ANSI X9.63, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 6.2.1 using the same domain parameters (*q, FR, a, b, [SEED], G, n, h*). Party U has ($d_{s,U}$, $Q_{s,U}$); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, each party generates an ephemeral key pair (d_e, Q_e) using the same domain parameters (*q, FR, a, b, [SEED], G, n, h*) that were used to generate the static key pair and sends the ephemeral public key Q_e to the other party. Party U generates ($d_{e,U}$, $Q_{e,U}$) and sends $Q_{e,U}$ to party V; party V generates ($d_{e,V}$, $Q_{e,V}$) and sends $Q_{e,V}$ to party U. Each party computes the shared secret *Z* as shown in Table 4, and then computes the shared keying material by invoking the key derivation function using *Z* (see Section 6.3).

Table 5: ANSI X9.63 Full Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(*q, FR a, b, [SEED], G, n, h*), $d_{e,U}$, $Q_{e,V}$, $d_{s,U}$, $Q_{s,V}$	(*q, FR, a, b, [SEED] G, n, h*), $d_{e,V}$, $Q_{e,U}$, $d_{s,V}$, $Q_{s,U}$
Computation	$(x_s, y_s) = hd_{s,U}Q_{s,V}$ $(x_e, y_e) = hd_{e,U}Q_{e,V}$ $Z_s = x_s$ $Z_e = x_e$	$(x_s, y_s) = hd_{s,V}Q_{s,U}$ $(x_e, y_e) = hd_{e,V}Q_{e,U}$ $Z_s = x_s$ $Z_e = x_e$
Derive Keying Material	Compute *kdf*(*Z,OtherInput*)using $Z = Z_e \| \| Z_s$	Compute *kdf*(*Z,OtherInput*)using $Z = Z_e \| \| Z_s$

7.1.1.3 MQV2, C(2,2,MQV,FF)

For the MQV2 scheme from ANSI X9.42, each party has a static key pair (*x, y*) that was previously generated as specified in Section 6.2.1 using the same domain parameters (*p, q, g*).

Party U has (x_U, y_U); party V has (x_V, y_V). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, each party generates an ephemeral key pair (r, t) using the same domain parameters (p, q, g) that were used to generate the static key pair and sends the ephemeral public key t to the other party. Party U generates (r_U, t_U) and sends t_U to party V; party V generates (r_V, t_V) and sends t_V to party U. Each party computes the shared secret Z as shown in Table 6, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 6: ANSI X9.42 MQV2 Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), x_U, y_V, r_U, t_U, t_V	(p, q, g), x_V, y_U, r_V, t_V, t_U
Computation	1. $w = \lceil \|\|q\|\|/2 \rceil$ 2. $t_U\text{¢} = (t_U \bmod 2^w) + 2^w$ 3. $S_U = (r_U + t_U\text{¢}\, x_U) \bmod q$ 4. $t_V\text{¢} = (t_V \bmod 2^w) + 2^w$ 5. $Z_{MQV} = \left(t_V y_V^{t_V'}\right)^{S_U} \bmod p$	1. $w = \lceil \|\|q\|\|/2 \rceil$ 2. $t_V\text{¢} = (t_V \bmod 2^w) + 2^w$ 3. $S_V = (r_V + t_V\text{¢}\, x_V) \bmod q$ 4. $t_U\text{¢} = (t_U \bmod 2^w) + 2^w$ 5. $Z_{MQV} = \left(t_U y_U^{t_U'}\right)^{S_V} \bmod p$.
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput*)using $Z = Z_{MQV}$	Compute *kdf*(*Z*,*OtherInput*)using $Z = Z_{MQV}$

7.1.1.4 Full MQV, C(2,2,MQV,EC)

For the Full MQV scheme from ANSI X9.63, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 6.2.1 using the same domain parameters (q, *FR*, a, b, *[SEED]*, G, n, h). Party U has ($d_{s,U}$, $Q_{s,U}$); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, each party generates an ephemeral key pair (d_e, Q_e) using the same domain parameters (q, *FR*, a, b, *[SEED]*, G, n, h) that were used to generate the static key pair and sends the ephemeral public key Q_e to the other party. Party U generates ($d_{e,U}$, $Q_{e,U}$) and

sends $Q_{e,U}$ to party V; party V generates ($d_{e,V}$, $Q_{e,V}$) and sends $Q_{e,V}$ to party U. Each party computes the shared secret Z as shown in Table 7, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 7 : ANSI X9.63 Full MQV Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(q, *FR* a, b, *[SEED]*, G, n, h), $d_{e,U}$, $Q_{e,V}$, $d_{s,U}$, $Q_{e,U}$, $Q_{s,V}$	(q, *FR*, a, b, *[SEED]* G, n, h), $d_{e,V}$, $Q_{e,U}$, $d_{s,V}$, $Q_{e,V}$, $Q_{s,U}$
Computation	1. $implicitsig_U = (d_{e,U} + avf(Q_{e,U})d_{s,U}) \bmod n$ 2. $(x, y) = h \times implicitsig_U \times (Q_{e,V} + avf(Q_{e,V})Q_{s,V})$ $Z = x$	1. $implicitsig_V = (d_{e,V} + avf(Q_{e,V})d_{s,V}) \bmod n$ 2. $(x, y) = h \times implicitsig_V \times (Q_{e,U} + avf(Q_{e,U})Q_{s,U})$ 3. $Z = x$
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput*)using Z	Compute *kdf*(*Z*,*OtherInput*)using Z

7.1.2 Each party generates an ephemeral key pair; no static keys are used: C(2,0)

For this category, only Diffie-Hellman schemes are specified. Each party generates ephemeral key pairs with the same domain parameters. The two parties exchange ephemeral public keys and then compute the shared secret. The keying material is derived using the shared secret (see Figure 2).

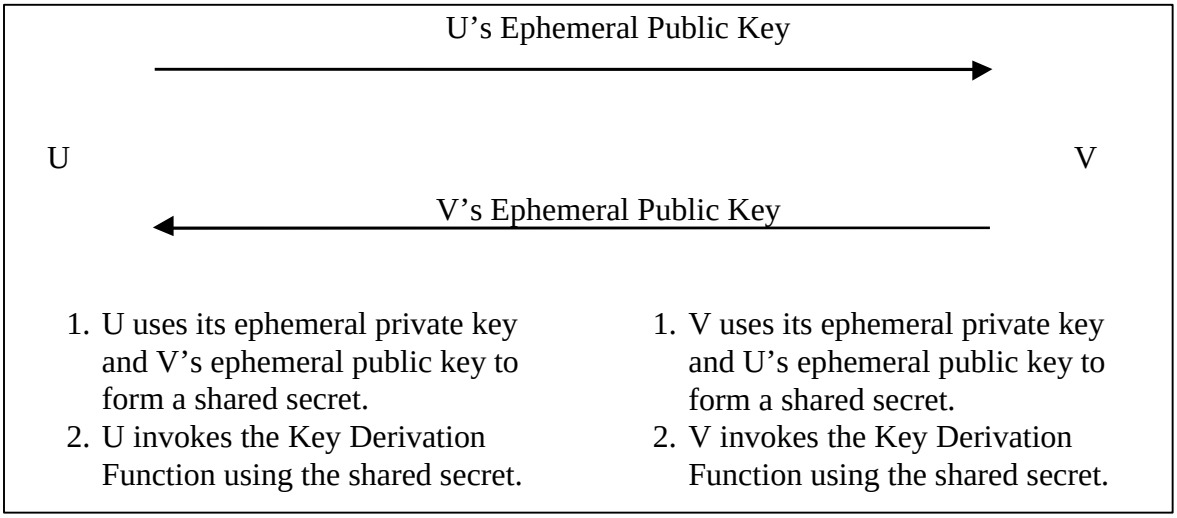

Figure 2: General protocol when each party generates ephemeral key pairs; no static keys are used

7.1.2.1 dhEphem, C(2,0,DH,FF)

In this scheme from ANSI X9.42, each party generates an ephemeral key pair (r, t) as specified in Section 6.2.1 using the same domain parameters (p, q, g) and sends the ephemeral public key t to the other party. Each party computes a shared secret Z as shown in Table 8. The shared keying material is computed by invoking the key derivation function using Z (see Section 6.3).

Table 8: ANSI X9.42 dhEphem Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	N/A
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	1. Ephemeral private key r_V 2. Ephemeral public key t_V
Input	(p, q, g), r_U, t_V	(p, q, g), r_V, t_U
Computation	$Z_e = t_V^{r_U} \bmod p$	$Z_e = t_U^{r_V} \bmod p$
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput* using $Z = Z_e$	Compute *kdf*(*Z*,*OtherInput* using $Z = Z_e$

7.1.2.2 Ephemeral Unified Model, C(2,0,DH,EC)

In this scheme from ANSI X9.63, each party generates an ephemeral key pair (d_e, Q_e) as specified in Section 6.2.1 using the domain parameters (q, *FR*, a, b, [*SEED*], G, n, h) and sends the ephemeral public key Q_e to the other party. Each party calculates a shared secret Z as shown in Table 9. The shared keying material is computed by invoking the key derivation function using Z (see Section 6.3).

Table 9: ANSI X9.63 Ephemeral Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	N/A
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	1. Ephemeral private key $d_{e,V}$ 2. Ephemeral public key $Q_{e,V}$
Input	(q, *FR* a, b, *[SEED]*, G, n, h), $d_{e,U}$, $Q_{e,V}$	(q, *FR*, a, b, *[SEED]* G, n, h), d_{eV}, $Q_{e,U}$
Computation	$(x_e, y_e) = hd_{e,U}Q_{e,V}$ $Z_e = x_e$	$(x_e, y_e) = hd_{e,V}Q_{e,U}$ $Z_e = x_e$

Derive Keying Material	Compute *kdf*(*Z,OtherInput* using $Z = Z_e$	Compute *kdf*(*Z,OtherInput* using $Z = Z_e$

7.2 One Party Participation (client/server, store-and-forward, 1-way), C(1)

In this category, the parties participating in a key agreement perform different calculations to determine the shared secret, depending on whether or not they initiate the key agreement process. Let party U serve as the initiator, and party V serve as the responder. Only the initiator (party U) generates an ephemeral key pair.

This category consists of two subcategories that are determined by the possession of static key pairs by the parties. In the first subcategory, both the initiator and the responder have static key pairs, and the initiator also generates an ephemeral key pair (see Section 7.2.1). In the second subcategory, the initiator generates an ephemeral key pair, but has no static key pair; the responder has only a static key pair (see Section 7.2.2).

7.2.1 Initiator has a static key pair and generates an ephemeral key pair; Responder has a static key pair, C(1,2)

For these schemes, party U (the initiator) uses both static and ephemeral private/public key pairs. Party V (the responder) uses only a static private/public key pair. Party U and party V obtain each other's static public keys in a trusted manner. Party U also sends its ephemeral public key to party V. A shared secret is generated by both parties using the available static and ephemeral keys. The shared keying material is derived using the shared secret (see Figure 3).

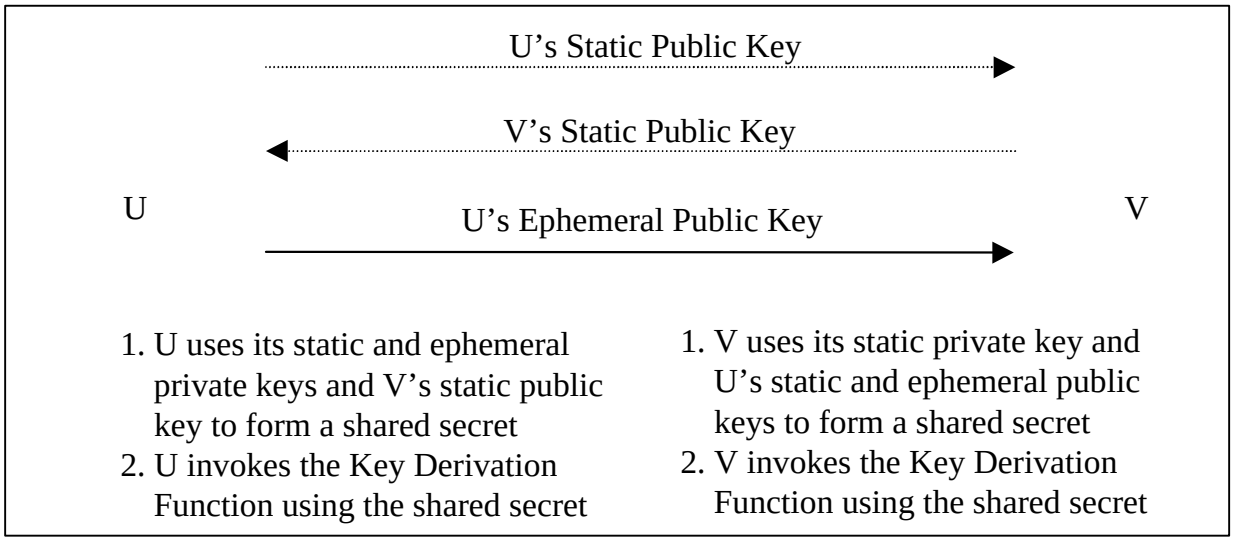

Figure 3: General protocol when the initiator has both static and ephemeral key pairs, and the responder has only a static key pair

7.2.1.1 dhHybridOneFlow, C(1,2,DH,FF)

In this scheme from ANSI X9.42, each party has a static key pair (*x*, *y*) that was previously generated as specified in Section 6.2.1 using the same domain parameters (*p, q, g*). Party U has

(x_U, y_U); party V has (x_V, y_V). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters (p, q, g) that were used to generate the static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret Z as shown in Table 10, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 10: ANSI X9.42 dhHybridOneFlow Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A
Input	(p, q, g), x_U, r_U, y_V	(p, q, g), x_V, y_U, t_U
Computation	$Z_s = y_V^{x_U} \bmod p$ $Z_e = y_V^{r_U} \bmod p$	$Z_s = y_U^{x_V} \bmod p$ $Z_e = t_U^{x_V} \bmod p$
Derive Keying Material	Compute *kdf*(*Z,OtherInput* using $Z = Z_e \| Z_s$	Compute *kdf*(*Z,OtherInput* using $Z = Z_e \| Z_s$

7.2.1.2 1-Pass Unified Model, C(1,2,DH,EC)

In this scheme from ANSI X9.63, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 6.2.1 using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*) that were used to generate the static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret Z as shown in Table 11, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 11: ANSI X9.63 1-Pass Unified Model Key Agreement Scheme

	Party U	Party V
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(q, FR, a, b, [$SEED$], G, n, h), $d_{s,U}$, $d_{e,U}$, $Q_{s,V}$	(q, FR, a, b, [$SEED$], G, n, h), $d_{s,V}$, $Q_{s,U}$, $Q_{e,U}$
Computation	$(x_s, y_s) = h\, d_{s,U}\, Q_{s,V}$ $(x_e, y_e) = h\, d_{e,U}\, Q_{s,V}$ $Z_s = x_s$ $Z_e = x_e$	$(x_s, y_s) = h\, d_{s,V}\, Q_{s,U}$ $(x_e, y_e) = h\, d_{s,V}\, Q_{e,U}$ $Z_s = x_s$ $Z_e = x_e$
Derive Keying Material	Compute *kdf*(Z,*OtherInput* using $Z = Z_e \,\|\|\, Z_s$	Compute *kdf*(Z,*OtherInput* using $Z = Z_e \,\|\|\, Z_s$

7.2.1.3 MQV1, C(1,2,MQV,FF)

For the MQV1 scheme from ANSI X9.42, each party has a static key pair (x, y) that was previously generated as specified in Section 6.2.1 using the same domain parameters (p, q, g). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters (p, q, g) that were used to generate the static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret Z as shown in Table 12, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 12: ANSI X9.42 MQV1 Key Agreement Scheme

	Party U	Party V
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A

Input	(p, q, g), x_U, y_V, r_U, t_U	(p, q, g), x_V, y_U, t_U
Computation	1. $w = \lceil \|q\|/2 \rceil$ 2. $t_U' = (t_U \bmod 2^w) + 2^w$ 3. $S_U = (r_U + t_U' x_U) \bmod q$ 4. $y_V' = (y_V \bmod 2^w) + 2^w$ 5. $Z_{MQV} = \left(y_V y_V^{y_V'}\right)^{S_U} \bmod p$	1. $w = \lceil \|q\|/2 \rceil$ 2. $y_V' = (y_V \bmod 2^w) + 2^w$ 3. $S_V = (x_V + y_V' x_V) \bmod q$ 4. $t_U' = (t_U \bmod 2^w) + 2^w$ 5. $Z_{MQV} = \left(t_U y_U^{t_U'}\right)^{S_V} \bmod p$
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput* using $Z = Z_{MQV}$	Compute *kdf*(*Z*,*OtherInput* using $Z = Z_{MQV}$

7.2.1.4 1-Pass MQV, C(1,2,MQV,EC)

For the 1-Pass MQV scheme from ANSI X9.63, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 6.2.1 using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **must** obtain the other party's static public key in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*) that were used to generate the static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret *Z* as shown in Table 13, and then computes the shared keying material by invoking the key derivation function using *Z* (see Section 6.3).

Table 13: ANSI X9.63 1-Pass MQV Model Key Agreement Scheme

	Party U	Party V
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(q, FR, a, b, [$SEED$], G, n, h), $d_{e,U}$, $d_{s,U}$, $Q_{e,U}$, $Q_{s,V}$	(q, FR, a, b, [$SEED$], G, n, h), $d_{s,V}$, $Q_{s,V}$, $Q_{e,U}$, $Q_{s,U}$
Computation	1. $implicitsig_U = (d_{e,U} + avf(Q_{e,U})d_{s,U}) \bmod n$ 2. $(x, y) = h \times implicitsig_U \times (Q_{s,V} + avf(Q_{s,V})\, Q_{s,V})$ 3. $Z = x$	1. $implicitsig_V = (d_{s,V} + avf(Q_{s,V})d_{s,V}) \bmod n$ 2. $(x, y) = h \times implicitsig_V \times (Q_{e,U} + avf(Q_{e,U})\, Q_{s,U})$ $Z = x$
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput* using *Z*	Compute *kdf*(*Z*,*OtherInput* using *Z*

7.2.2 Initiator generates only an ephemeral key pair; Responder has only a static key pair, C(1,1)

For these schemes, Party U generates an ephemeral key pair, but has no static key pair; party V has only a static key pair. Party U obtains party V's static public key in a trusted manner and sends its ephemeral public key to Party V. The parties compute a shared secret using their private keys and the other party's public key. Each party uses the shared secret to derive keying material (see Figure 4).

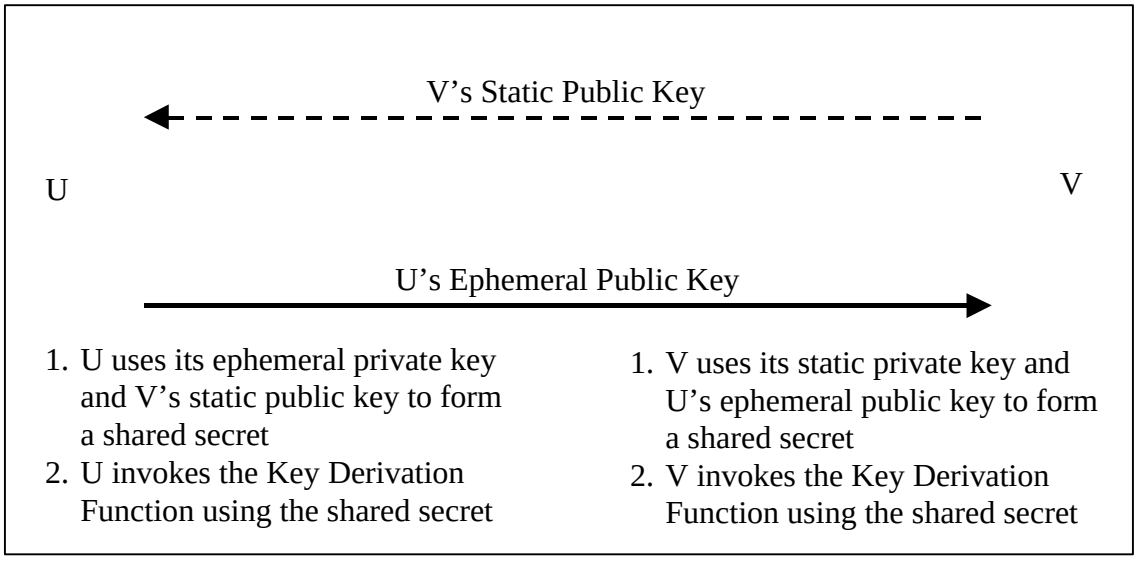

Figure 4: General protocol when the Initiator has only an ephemeral key pair, and the Responder has only a static key pair

7.2.2.1 dhOneFlow, C(1,1,DH,FF)

In this scheme from ANSI X9.42, party V has a static key pair (x_V, y_V) that was previously generated as specified in Section 6.2.1 using domain parameters (p, q, g). Party U **must** obtain party V's static public key in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair (r_U, t_U) using the same domain parameters (p, q, g) that were used to generate party V's static key pair and sends the ephemeral public key t_U to party V (the responder). Each party computes the shared secret Z as shown in Table 14, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 14 : ANSI X9.42 dhOneFlow Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	1. Ephemeral private key r_U 2. Ephemeral public key t_U	N/A
Input	(p, q, g), r_U, y_V	(p, q, g), x_V, t_U
Computation	$Z_e = y_V^{r_U} \bmod p$	$Z_e = t_U^{x_V} \bmod p$

Derive Keying Material	Compute *kdf*(*Z,OtherInput* using $Z = Z_e$	Compute *kdf*(*Z,OtherInput* using $Z = Z_e$

7.2.2.2 1-Pass Diffie-Hellman, C(1,1,DH,EC)

In this scheme from ANSI X9.63, party V has a static key pair ($d_{s,V}$, $Q_{s,V}$) that was previously generated as specified in Section 6.2.1 using domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*). Party U **must** obtain party V's static public key ($Q_{s,V}$) in a trusted manner.

During the key agreement process, party U (the initiator) generates an ephemeral key pair ($d_{e,U}$, $Q_{e,U}$) using the same domain parameters (*q, FR, a, b,* [*SEED*], *G, n, h*) that were used to generate party V's static key pair and sends the ephemeral public key $Q_{e,U}$ to party V (the responder). Each party computes the shared secret *Z* as shown in Table 15, and then computes the shared keying material by invoking the key derivation function using *Z* (see Section 6.3).

Table 15: ANSI X9.63 1-Pass Diffie-Hellman Model Key Agreement Scheme

	Party U	**Party V**
Static Data	N/A	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	1. Ephemeral private key $d_{e,U}$ 2. Ephemeral public key $Q_{e,U}$	N/A
Input	(*q, FR, a, b,* [*SEED*], *G, n, h*), $d_{e,U}$, $Q_{s,V}$	(*q, FR, a, b,* [*SEED*], *G, n, h*), d_{sV}, $Q_{e,U}$
Computation	$(x, y) = h\, d_{e,U}\, Q_{s,V}$ $Z = x$	$(x, y) = h\, d_{s,V}\, Q_{e,U}$ $Z = x$
Derive Keying Material	Compute *kdf*(*Z,OtherInput* using *Z*	Compute *kdf*(*Z,OtherInput* using *Z*

7.3 Static keys only, C(0)

In this category, each party has only static key pairs that have been generated using the same domain parameters. Each party obtains the other party's static public keys and calculates the shared secret by using their own static private key and the other party's static public key. Keying material is derived using the key derivation function and the shared secret (see Figure 5).

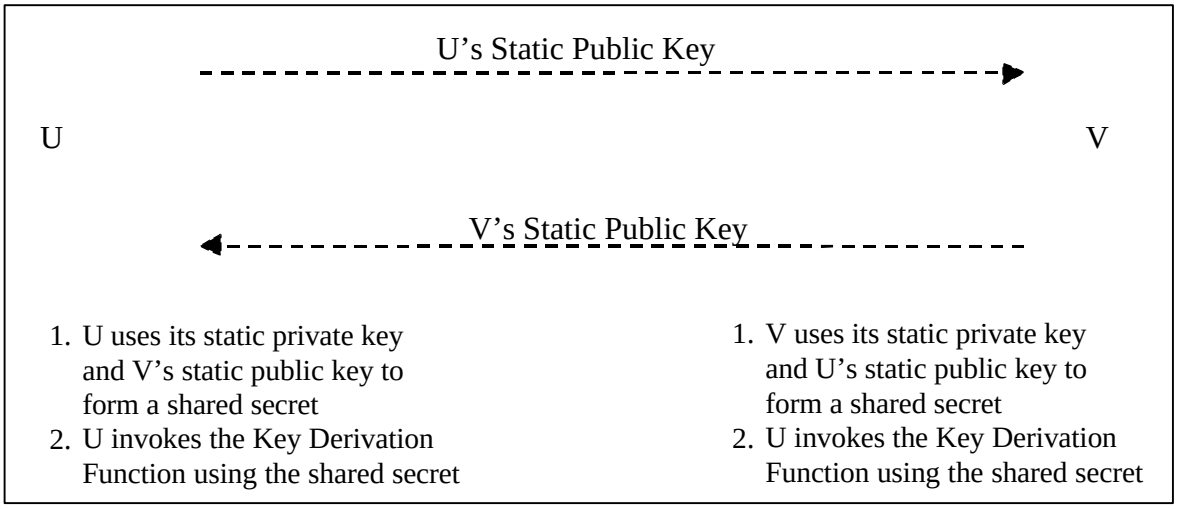

Figure 5: Each party has only a static key pair

7.3.1 dhStatic, C(0,2,DH,FF)

In this scheme from ANSI X9.42, each party has a static key pair (x, y) that was previously generated as specified in Section 6.2.1 using the same domain parameters (p, q, g). Party U has (x_U, y_U); party V has (x_V, y_V). Each party **must** obtain the other party's static public key in a trusted manner.

Each party computes the shared secret Z as shown in Table 16, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 16: ANSI X9.42 dhStatic Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key x_U 2. Static public key y_U	1. Static private key x_V 2. Static public key y_V
Ephemeral Data	N/A	N/A
Input	(p, q, g), x_U, y_V	(p, q, g), x_V, y_U
Computation	$Z_s = y_V^{x_U} \bmod p$	$Z_s = y_U^{x_V} \bmod p$
Derive Keying Material	Compute *kdf*(*Z*,*OtherInput* using Z = Z_s	Compute *kdf*(*Z*,*OtherInput* using Z = Z_s

7.3.2 Static Unified Model, C(0,2,DH,EC)

In this scheme from ANSI X9.63, each party has a static key pair (d_s, Q_s) that was previously generated as specified in Section 6.2.1 using the same domain parameters (q, FR, a, b, [$SEED$], G, n, h). Party U has (d_{sU}, Q_{sU}); party V has ($d_{s,V}$, $Q_{s,V}$). Each party **must** obtain the other party's static public key in a trusted manner.

Each party computes the shared secret Z as shown in Table 17, and then computes the shared keying material by invoking the key derivation function using Z (see Section 6.3).

Table 17: ANSI X9.63 Static Unified Model Key Agreement Scheme

	Party U	**Party V**
Static Data	1. Static private key $d_{s,U}$ 2. Static public key $Q_{s,U}$	1. Static private key $d_{s,V}$ 2. Static public key $Q_{s,V}$
Ephemeral Data	N/A	N/A
Input	(q, FR, a, b, [$SEED$], G, n, h), $d_{s,U}$, $Q_{s,V}$	(q, FR, a, b, [$SEED$], G, n, h), $d_{s,V}$, $Q_{s,U}$
Computation	$(x_s, y_s) = hd_{s,U}Q_{s,V}$ $Z_s = x_s$	$(x_s, y_s) = hd_{s,V}Q_{s,U}$ $Z_s = x_s$
Derive Keying Material	Compute $kdf(Z,OtherInput$ using $Z = Z_s$	Compute $kdf(Z,OtherInput$ using $Z = Z_s$

9. Key Transport

[To be addressed]

10. Keys Derived from a "Master Key"

[Suggestions on this section are welcome]

11. Key Recovery

For some applications, the keying material used to protect data may need to be recovered (e.g., the normal reference copy of the keying material is lost or corrupted). In this case, either the keying material or sufficient information to reconstruct the keying material needs to be available (e.g., the keys, domain parameters and the scheme used to perform the key establishment process).

Keys used during the key establishment process must be handled in accordance with the following:

- Static key pairs may be saved (see the Key Management Guideline document [8] for required protections); static public keys could be saved, for example, in public key certificates.

- Ephemeral public keys may be saved.
- Ephemeral private keys **must not** be recoverable, and, therefore, must not be saved.

[Note: This implies that schemes where both parties generate ephemeral key pairs (see Section 7.1) cannot be recoverable by reconstruction of the keying material. For those schemes where only the initiator generates an ephemeral key pair (see Section 7.2), only the responder can recover the keying material by reconstruction.]

[Note: Text may need to be added about recovering the key derived from a master key.]

General guidance on key recovery and the protections required for each type of key is provided in the Key Management Guideline document [8].

12. Implementation Validation

Implementations of schemes in the final schemes document **must** be tested in order to claim compliance with this workshop document. Information on NIST's testing program is available at http://csrc.nist.gov/cryptval.

Appendix A: Secure Implementation

Appendix B: Security Considerations

Appendix C: Examples

Appendix E: References

[1] FIPS 140-2, *Security requirements for Cryptographic Modules*, May 25, 2001.

[2] FIPS 180-2 (Draft), *Secure Hash Standard*, [Insert data].

[3] FIPS 186-3 (Draft), *Digital Signature Standard*, [Insert date].

[4] FIPS 196, *Entity Authentication Using Public Key Cryptography*, February, 1997.

[5] FIPS 197, *Advanced Encryption Standard*, [Insert date].

[6] FIPS 198 (Draft), *The Keyed-Hash Message Authentication Code (HMAC)*, [Insert date].

[7] SP 800-XX, *Recommendation for Block Cipher Modes of Operation*, [Insert date].

[8] SP 800-XX, *Key Management Guideline*, [Insert date].

[9] ANSI X9.42 -2001, *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*

[10] ANSI X9.44 (Draft), *Public Key Cryptography for the Financial Services Industry: Agreement and Key Transport Using Factoring-Based Cryptography*

[11] ANSI X9.63 (Draft), *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Key Cryptography*

[12] ANSI X9.80-2001, *Prime number Generation, Primality Testing and Primality Certificates*

[13] ANSI X9.82 (Draft), *Random Bit Generation*

[14] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997